TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanksystemen* im SoSe18

Alexander van Renen, Maximilian E. Schüle (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss18/impldb/

Blatt Nr. 07

Hausaufgabe 1

Zeigen Sie die weiteren Phasen des Apriori-Algorithmus für unser Beispiel in Abbildung 1 (hier ist lediglich bis inkl. 2. Phase dargestellt). Damit eine Menge von Produkten ein Frequentitemset ist, muss sie in mindestens 3/5 aller Verkäufe enthalten sein, d.h. $minsupp = s_0 = 3/5$. Gehen Sie für die Assoziationsregeln von einer minimalen Konfidenz von $k_0 = 0$ aus und berechnen Sie die Konfidenz der Assoziationsregel {Drucker}⇒ {Papier, Toner}.

VerkaufsTransaktionen

TransID	Produkt
111	Drucker
111	Papier
111	PC
111	Toner
222	PC
222	Scanner
333	Drucker
333	Papier
333	Toner
444	Drucker
444	PC
555	Drucker
555	Papier
555	PC
555	Scanner
555	Toner

Zwischenergebnisse

FI-Kandidat	Anzahl
{Drucker}	4
{Papier}	3
{PC}	4
{Scanner}	2
{Toner}	3
{Drucker, Papier}	3
{Drucker, PC}	3
{*Drucker, Scanner*}	
{Drucker, Toner}	3
{Papier, PC}	2
{*Papier, Scanner*}	
{Papier, Toner}	3
{*PC, Scanner*}	
{PC, Toner}	2
{*Scanner, Toner*}	

Abbildung 1: Ausgangssituation für den Apriori-Algorithmus

Vgl. Übungsbuch 17.6. Frequentitemsets sind alle nicht gestrichenen (wegen zu geringem Supports) bzw. nicht kursiv gesetzten (wegen nicht häufig auftretender Teilmenge).

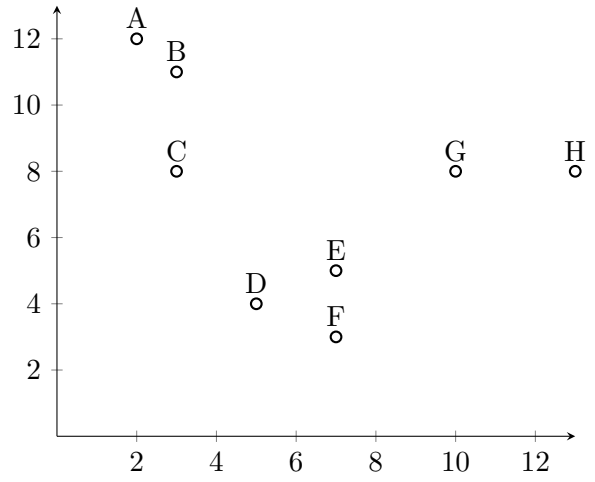
Iteration	Item-Menge X	$\sigma(X)$	s(X)
1	{Drucker}	4	4/5
1	{Papier}	3	3/5
1	{PC}	4	4/5
1	~~{Scanner}~~	2	2/5
1	{Toner}	3	3/5
2	{Drucker, Papier}	3	3/5
2	{Drucker, PC}	3	3/5
2	{*Drucker, Scanner*}		
2	{Drucker, Toner}	3	3/5
2	~~{Papier, PC}~~	2	2/5
2	{*Papier, Scanner*}		
2	{Papier, Toner}	3	3/5
2	{*PC, Scanner*}		
2	~~{PC, Toner}~~	2	2/5
2	{*Scanner, Toner*}		
3	{*Drucker, Papier, PC*}		
3	{Drucker, Papier, Toner}	3	3/5
3	{*Drucker, PC, Toner*}		
3	{*Papier, PC, Toner*}		

Der Vollständigkeit halber im Nachfolgenden alle möglichen Assoziationsregeln.

Item-Menge X	$\sigma(X)$	$s(X)$	$c(X)$
$\emptyset \Rightarrow$ {Drucker}	4	4/5	4/5
$\emptyset \Rightarrow$ {Papier}	3	3/5	3/5
$\emptyset \Rightarrow$ {PC}	4	4/5	4/5
$\emptyset \Rightarrow$ {Toner}	3	3/5	3/5
$\emptyset \Rightarrow$ {Drucker, Papier}	3	3/5	3/5
{Drucker}$\Rightarrow$ {Papier}	3	3/5	3/4
{Papier}$\Rightarrow$ {Drucker}	3	3/5	3/3
$\emptyset \Rightarrow$ {Drucker, PC}	3	3/5	3/5
{Drucker}$\Rightarrow$ {PC}	3	3/5	3/4
{PC}$\Rightarrow$ {Drucker}	3	3/5	3/4
$\emptyset \Rightarrow$ {Drucker, Toner}	3	3/5	3/5
{Drucker}$\Rightarrow$ {Toner}	3	3/5	3/4
{Toner}$\Rightarrow$ {Drucker}	3	3/5	3/3
$\emptyset \Rightarrow$ {Papier, Toner}	3	3/5	3/5
{Papier}$\Rightarrow$ {Toner}	3	3/5	3/3
{Toner}$\Rightarrow$ {Papier}	3	3/5	3/3
$\emptyset \Rightarrow$ {Drucker, Papier, Toner}	3	3/5	3/5
{Drucker}$\Rightarrow$ {Papier, Toner}	3	3/5	3/4
{Drucker, Papier}$\Rightarrow$ {Toner}	3	3/5	3/3
{Drucker, Toner}$\Rightarrow$ {Papier}	3	3/5	3/3
{Papier}$\Rightarrow$ {Drucker, Toner}	3	3/5	3/3
{Papier, Toner}$\Rightarrow$ {Drucker}	3	3/5	3/3
{Toner}$\Rightarrow$ {Drucker, Papier}	3	3/5	3/3

Hausaufgabe 2

Folgende Datenpunkte im euklidischen Raum seien gegeben:



Punkt	X	Y
A	2	12
B	3	11
C	3	8
D	5	4
E	7	5
F	7	3
G	10	8
H	13	8

Clustern Sie die Punkte mithilfe des *k-means*-Verfahren in 3 Cluster. Nutzen Sie als initiale Clusterzentren die Werte A, B und C. Wenn ein Punkt zu mehreren Clustern die gleiche Distanz hat, wird er dem Cluster der näher am Nullpunkt liegt zugeordnet. Geben Sie für jede Iteration jeweils die Zuordnung und die Mittelpunkte der Cluster an.

Eine Iteration des K-Means-Algorithmus kann wie folgt ausgewertet werden:

```
with points(id,x,y) as (
        VALUES ('A', 2, 12), ('B', 3, 11), ('C', 3,8), ('D', 5,4),
        ('E',7,5),('F',7,3),('G',10,8),('H',13,8)
),
clusters_0(cid,x,y) as (
  VALUES ('1', 2, 12), ('2', 3, 11), ('3', 3,8)
),
clusters_1(cid, x,y, count) as (
  select cid, avg(px), avg(py), count(*) from (
    select cid, p.x as px, p.y as py, rank() OVER (
      partition by p.id
      order by (p.x-c.x)*(p.x-c.x)+(p.y-c.y)*(p.y-c.y) asc,
        (c.x*c.x+c.y*c.y) asc)
    from points p, clusters_0 c
  ) x
  where x.rank=1
  group by cid
)
```

Die Clusterzentren können mit folgender Abfrage ausgegeben werden

```
select * from clusters_1
```

Die Zuordnung kann mit folgender Abfrage ausgewertet werden

```
select cid,pid from (
    select cid, p.id as pid, rank() OVER (
        partition by p.id
        order by (p.x-c.x)*(p.x-c.x)+(p.y-c.y)*(p.y-c.y) asc,
            (c.x*c.x+c.y*c.y) asc)
    from points p, clusters_1 c
) x
where x.rank=1
```

Hausaufgabe 3

Wie ändert sich die Bedeutung des Redo-Log und Undo-Log in Hauptspeicherdatenbanken im Vergleich zu klassischen Datenbanken? Wo werden sie gespeichert?

Da die Daten nicht mehr auf der Festplatte gespeichert werden, müssen sie (falls es keine Snapshots gibt) bei einem Neustart komplett auf Basis des Redo-Logs wiederhergestellt werden. Da nie die Daten einer nicht comitteten Transaktion auf der Platte landen wird die Undo-Log nur benötigt um im Fall eines Abort die Änderungen der Transaktion zurückzusetzen. Außerdem reicht es aus die Redo Log Daten erst beim Commit zu schreiben. Aus diesem Grund sind die Daten der Undo-Log nur zur Laufzeit der Transaktion notwendig und müssen nicht auf die Festplatte geschrieben werden. Während der Wiederherstellung der Datenbank können dann einfach alle Redo Log Einträge wiederhergestellt werden.

Hausaufgabe 4

HyPer schafft 120.000 Transaktionen pro Sekunde. Pro Transaktion werden 120 Byte in die Log geschrieben. Berechnen Sie den benötigten Durchsatz zum Schreiben der Log.

Die Datenbank läuft für einen Monat und stürzt dann ab. Es wurde kein Snapshot erstellt. Berechnen Sie die Recoveryzeit. Gehen Sie davon aus, dass die Recovery durch die

Festplatte limitiert ist (100 MiB / s). Wieviel Log Einträge werden pro Sekunde recovert?

Durchsatz = 120.000 * 120 = 14400000 = 13,7 MiB/s.

LogEinträge =120.000*60*60*24*30=311040000000

LogGröße = LogEinträge*120 = 33,95 TiB

RecoveryZeit = 4,12 Tage

RecoveryDurchsatz = 873813 Tx / s.

Hausaufgabe 5

In traditionellen Datenbanksystemen sind die Festplatte und der Buffermanager oft der Hauptgrund für Performanceengpässe. Wie ändert sich dies in Hauptspeicherdatenbanken, wo sind die neuen Flaschenhälse? Unterscheiden Sie auch zwischen Analytischen und Transaktionalen Workloads.

Der Unterschied zwischen traditionellen Datenbanksystemen und Hauptspeicherdatenbanke ist, dass wir in der Speicherhierarchie ein paar Stufen nach oben gehen. Hauptspeicher ist teurer, aber gleichzeitig auch schneller und hat eine geringere Latenz. Genauso wichtig ist aber auch, dass die Daten nun alle in einem Adressraum liegen. Bei der Nutzung von Festplatten muss das Datenbanksystem explizit die Daten von der Festplatte in den Speicher laden. Beim Hauptspeicher ist der Wechsel zwischen RAM, L3 und L1 Cache transparent für das System. Das heißt ein Buffermanager wird nicht mehr benötigt. Auch wenn der Wechsel zwischen den verschiedenen Hauptspeicherhierarchien für das System nicht explizit sichtbar ist, so ist es doch in der Performance bemerkbar. Der Latenzunterschied zwischen RAM und L3 ist ähnlich groß wie zwischen Festplatten und RAM. Die Datenbank muss nun so strukturiert werden, dass möglichst viele Operationen auf Daten in den schnelleren Speicherschichten ausgeführt werden. Ein weiterer neuerer Flaschenhals sind die Lockingverfahren. Im Vergleich zu einfachen Operationen wie Lesen, Schreiben, Addition, etc. ist ein Lock zu erstellen viel teurer. Viele Hauptspeichersystem versuchen daher Locks zu vermeiden. Ein Problem, dass hauptsächlich nur Transaktionale Workloads betrifft ist, dass beim Ändern von Daten die Änderung immer noch persistiert werden muss (Logging). Es reicht nicht aus, die Daten nur im Hauptspeicher zu halten, da diese dann nach einem Systemabsturz verloren wären. Das Schreiben auf Festplatte ist selbst mit SSDs noch wesentlich teurer als Änderungen im Hauptspeicher vorzunehmen. Auch ein Problem in Transaktionale Workloads ist die Annahme der Anfragen. Transaktionale Anfragen sind typischerweiße sehr schnell. Ein einzelner Rechner kann sehr einfach mehrere Hunderttausend Anfragen verarbeiten. Hier ist die Netzwerklatenz auch wesentlich größer als die Verarbeitungszeit der Anfragen. Daher kann ein einzelner Client die Datenbank garnicht auslasten wenn er jede Anfrage einzeln abschickt.

Hausaufgabe 6

In (pseudo) Java kann eine 'Row-Store-artige' Datenstruktur wie folgt angelegt werden:

```
class Tuple {
  int MatrNr;
  String Name;
  int Semester;
}
Tuple data[]=new Tuple[10000];
```

Notieren Sie, wie die Daten in Form eines Column Stores gehalten werden können in (pseudo) Java.

Erklären Sie Ihrem Tutor, welche Vor- und Nachteile Row- und Column Stores jeweils haben. Was würden Sie für Amazons Webseite verwenden? Was verwenden Sie für die Controlling Datenbank?

```
int MatrNrs[]=new int[10000];
String Names[]=new String[10000];
int Semesters[]=new int[10000];
```

Row Store: Besser, wenn tendenziell viele Attribute des Tupels benutzt werden. Schlecht, wenn nur auf einen Bruchteil des Tupel zugegriffen wird, da viel mehr Daten geladen werden müssen und die Lokalität in der gesamten Speicherhierarchie dann schlechter ist.
Column Store defakto umgekehrt.
Im Schnitt verwendet man heute Row Stores für transaktionale Daten, Column Stores für analytische Daten. Hiervon kann abgewichen werden. Zu bedenken ist, welche Probleme entstehen können, wenn die Anwendungslogik nicht sinnvoll mit dem Datenbanksystem umgeht, beispielsweise weil immer alle Daten (`SELECT * FROM`) und nicht nur die benötigten ausgelesen werden.

Hausaufgabe 7

In Hauptspeicherdatenbanken ist die Geschwindigkeit oft durch Limitierungen des Speichersystems begrenzt. Analysieren sie dazu folgende Fragestellungen:

1. Was versteht man unter NUMA und welche Schichten gibt es in der Speicherhierarchie? Geben Sie zu jeder Schicht auch die Zugriffszeiten und Bandbreite an.
2. Was bedeuten die Begriffe *Cacheline* und *Seite*. Auf welcher Schicht sind diese jeweils relevant?

0.0.1 NUMA

Non-Uniform Memory Access oder kurz NUMA ist eine Computer-Speicher-Architektur für Multiprozessorsysteme, bei denen jeder Prozessor einen eigenen, lokalen Speicher hat, aber anderen Prozessoren über einen gemeinsamen Adressraum direkten Zugriff darauf gewährt (Distributed Shared Memory). Die Speicherzugriffszeiten in einem solchen Verbund hängen daher davon ab, ob sich eine Speicheradresse im lokalen oder im fremden Speicher befindet (Wikipedia).
Bei der Programmierung von Programmen ist diese Trennung nicht sichtbar, bei der Ausführung kann diese aber zu großen Geschwindigkeitsschwankungen führen.

0.0.2 Speicherhierarchie

Register <1ns Zugriffszeit

L1 2ns Zugriffszeit, Bandbreite: 16 byte/Takt, 44 GBytes/s *pro CPU Kern*

L2 20ns Zugriffszeit, Bandbreite: 16 byte/Takt, 44 GBytes/s *pro CPU Kern*

Hauptspeicher 200ns Zugriffszeit, Bandbreite: 50 GByte/s to local socket, 16 GByte/s to remote socket

Festplatte 5ms Zugriffszeit, Bandbreite: 1GByte/s

Wichtig ist vor allem ein Gefühl für die Größenordnungen.

0.0.3 Zugriffsgranularität

Wenn der Prozessor auf einen Datenwert zugreift, müssen die entsprechenden Daten aus dem Hauptspeicher bzw Cache geladen werden. Eine Cache-Line ist die kleinste Verwaltungseinheit innerhalb des Caches von Prozessoren. Die Zugriffe vom Cache-Speicher zur CPU oder zum Hauptspeicher erfolgen somit in einem einzigen, blockweisen Transfer. Falls z.B. auf eine Zahl in einem der Caches zugegriffen wird muss immer die gesamte Cacheline, ein Block von 64-Byte Größe, geladen werden. Genauso beim Schreiben: Selbst wenn nur ein einzelnes Byte verändert wurde muss immer die ganze Cacheline aktualisiert werden. Die Verwaltung des Hauptspeichers durchs Betriebssystem erfolgt in noch gröberen Blöcken, sogenannten Seiten. Eine Seite ist dabei typischerweiße 4 Kilobyte groß. Dies ist typischerweiße auch die minimale Granularität mit der Daten auf die Festplatte geschrieben werden können.